Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 47 (2015) 188 – 196

Procedia Computer Science

An Approach for Generating Minimal Test Cases for Regression Testing

Sapna P G, Arunkumar Balakrishnan

aCoimbatore Institute of Technology, Coimbatore 641 005, India

Abstract

One of the main objectives of regression testing is to test that the changed system works according to specification, at the same time optimizing the number of test cases by making it efficient and effective. This paper presents a black-box approach that uses Steiner Tree algorithm to generate a minimal test set to check functionality. Test cases are generated from specification represented using the Unified Modeling Language (UML). A set of terminals are given as input to the Steiner Tree algorithm with the graph G. The changed nodes are defined as terminal nodes to ensure inclusion in the test set. A minimal set of test cases is generated as an indicator to the effectiveness of the change. Initial results show that the method is applicable for quick testing to ensure that basic functionality works correctly.

Peer-review under responsibility of organizing committee of the Graph Algorithms, High Performance Implementations and Applications (ICGHIA2014)

Keywords: Regression Testing, Test case generation, Unified Modeling language, Specification,

1. Introduction

Regression testing is the process of testing a system to verify that changes incorporated work correctly and meets specified requirements. Hence, regression testing involves testing a set of features that could be affected due to

* Sapna P G Email:sapnapga@gmail.com

Peer-review under responsibility of organizing committee of the Graph Algorithms, High Performance Implementations and Applications (ICGHIA2014)
doi:10.1016/j.procs.2015.03.197

modification of a feature or function. The modification could have been caused by resolving a bug or by an enhancement [1][2][3].

Given the nature of the software project development cycle, regression testing is a difficult task due to time and cost constraints in terms of customers requiring delivery of the product, developers working to meet targets and managers trying to keep project costs under control. Various techniques presented in literature look at generating regression test cases in order to evaluate the efficacy of a bug fix. A simple way to regression testing is to retest-all but is expensive. Besides, the time constraints on a software project do not allow exhaustive testing. Most regression testing techniques are code based which is advantageous in case of unit testing but faces scalability issues as size of software increases.

The Unified Modeling Language (UML) is a widely accepted standard for modeling software systems [4][5]. It consists of a set of modeling concepts (primitives) to support and object oriented approach to software development. UML consists of a set of diagrams that model both static and dynamic behaviour of a system. Various aspects of the system are elaborated at different levels of abstraction using diagrams like use case diagram, class diagram, activity diagram, sequence diagram and state diagram. As the current work involves functionality level, activity diagram is used. Activity diagrams elaborate scenarios related to each use case. Each functionality (use case) is elaborated using an activity diagram. The diagram models the main, alternate and exception scenarios with reference to the functionality[UML]. The number of scenarios generated from activity diagrams using automated generation is exhaustive.

Test selection consists of selecting a subset of test cases in order to ensure that developed functionality, existing and modified work correctly without compromising on quality. Rothermel and Harrold [6][7] have formally defined the regression test selection(RTS) problem as follows: Let P be an application program and P' be a modified version of P. Let T be the test suite developed initially for testing P. An RTS technique aims to select a subset of test cases $T' \subseteq T$ to be executed on P', such that every error detected when P' is executed with T is also detected when P' is executed with T'.

The main objective of selecting a subset of test cases is to identify a small set of test cases that test functionality of the system at the same time not compromising on quality. In this work, the small set of test cases will be referred to as safe regression testing as it considers functionality to be tested along with parts of software that have been changed and will be called the minimal test set. Hence, the premise is that determining a minimal set of test cases that will exercise functionality considering changes before elaborately testing the functionality to confirm that all regression test cases pass will be more effective. Such a set of test cases will aid in two ways: functionality that pass the initial test cases can be tested further to ensure the changes have not introduced further defects. In case a functionality fails the minimal test cases, then it will be safe to suffice that the bug fix or enhancement has failed.

A lot of work is available for generating regression test cases both, white-box and black-box strategies. This work focuses on black-box testing where changes related to a functionality of a system are regression tested. Further, this work looks at selecting a minimum regression test set that acts as a precursor for elaborate regression testing. This minimal regression test set indicates those functionality that have defects and hence need rework before elaborate testing and those that may be elaborately tested as they have passed the minimal regression test. Given the high cost of software development and testing effort there is scope to introduce new ways of enhancing the testing process. In this direction, the paper proposes an approach that uses a Steiner tree algorithm [7][8] to create the minimum regression test set.

The algorithm creates a Steiner tree using nodes in an activity diagram for each functionality. The input is the activity diagram representing each use case (functionality) of the system. The activity diagram is viewed in the form of a Control Flow Graph (CFG), with weights assigned for the edges. Given the terminal and non-terminal nodes, a Steiner tree, T, is generated. This tree, T, is used to elaborate test cases that will constitute the minimal regression test set. The result of applying the algorithm is studied.

The following benefits are expected through application of the technique:
1. smaller subset of effective regression tests aid in faster defect detection
2. the minimal regression test set acts as a precursor for further testing

The rest of the paper is organized as follows: Section 2 present related work in the area of regression testing. The proposed technique for generation of minimal regression test set is given in Section 3. An example illustrates the approach in section 4. Section 5 concludes the work and discusses future scope.

2. Background

2.1 Regression Testing

Regression testing strategies work at two levels: white-box and black-box [11]. A majority of regression testing techniques are white-box testing strategies and are at the source-code level i.e. the techniques use the source code of the original and modified programs to define regression test cases [12][13]. A call graph, control flow graph or dependence graph based on the source code is used as the basis for determining regression test cases. The advantage of using white-box strategy is that the approach is fine-grained viz. changes at the code level are monitored. The disadvantage is the loss of functional perspective.

Specification based regression testing techniques [12] use specification in the form of requirements or high level design as the basis for regression testing. This is at the level of functionality. The effect of a change is studied by developing at the functional level[13]16]. Both of the approaches develop a change impact model in order to understand the effect of a change. Also, though both of the above approaches work at different levels, they complement each other[12] in identifying defects.

Regression test selection techniques have been used in [6][7][14] to select a subset of test cases. The objective is to check if the modified program has the same behaviour as the previous acceptable version of the program running on T, a set of test cases [5]. UML activity diagrams have been used for specification. In [22], the authors present a framework for analyzing regression test selection techniques. Four categories are included in the framework : inclusiveness, precision, efficiency and generality. The authors analyze different technique on the four factors. In [7], a CFG is constructed for a procedure or program and it modified version. The CFG is used to select tests that execute changed code from the original test suite. Risk based test selection for regression testing is the focus of work done by Chen et al [15]. Risk information pertaining to test cases is provided by test engineers using experience gained. Risk metrics are used to measure quality of the test suite. Risk is based on the cost of each test case(valued by both customer and test engineer) as well as severity. Risk exposure (RE), a product of cost and severity is taken as the basis for test selection. Scenarios that cover most critical cases are considered first.

2.2 UML and Activity Diagrams

UML is a widely accepted standard for modeling object oriented software systems. Being a semi-formal language, it is widely used to specify requirements and depict design of the software. UML provides diagrams to represent the static as well as the dynamic behaviour of a system. Class, component and deployment diagrams are used to represent the static behaviour of the system whereas activity, sequence and state diagrams are used to represent the dynamic behaviour. Scenarios repesent the sequence of events in a software system and defines a system's behaviour. Use Case, activity, sequence and collaboration diagrams can be used to represent scenarios. Each scenario can be said to represent a requirement goal of the system. In practice, generation of scenarios is mostly done manually making it labor-intensive and error-prone. Hence, there is a need to generate test scenarios to achieve test adequacy and to ensure software quality [17]. Each scenario is considered as a test case. In this regard, automation of test case generation gains importance.

An activity diagram consists of activities and transitions, showing the flow of control from activity to activity as shown in Figure 1 for the functionality 'Customer does Online Shopping'. Online customers can browse or search

items, view specific item, add an item to the shopping cart, view and update shopping cart and do checkout. Also, users can view the shopping cart at any time. Checkout is assumed to include user registration and login. Each action is represented using an activity construct and the flow of control from one activity to another, represented using arrows (edges). Activity diagrams can be used to model both sequential and concurrent activities. Also, an activity diagram can be viewed as a graph with nodes representing activities and edges labeled with transitions as shown in Figure 1. UML activity diagrams are developed using two types of nodes, namely, action nodes and control nodes as shown in Figure 1. Action nodes include Activity, CallBehaviourAction, SendSignal and AcceptEvent. Control nodes include InitialNode, FinalNode, Flow-Final, Decision, Merge, Fork and Join. Also, UML 2.1 superstructure describes several levels of activity modeling: Basic, Intermediate, Complete, Structured, complete-Structured and Extra-Structured activities [4]. Definition 1 below defines an activity diagram.

Definition 1: An Activity diagram D is a tuple,

$D = (A, T, F, J, R, a_I, a_F)$ where
$A = a_0, a_1, a2, ..., a_m$ is a finite set of activities (nodes),
$T = t_0, t_1, t_2, ..., t_n$ is a finite set of transitions (edges),
$F = f_1, f_2, ... f_k$ is a finite set of forks,
$J = j_1, j_2, ..., j_k$ is a finite set of joins,
$R \subset (A \times T) \cup (T \times A)$ is the flow relation,
a_0 is the initial state, and
a_m is the final state.

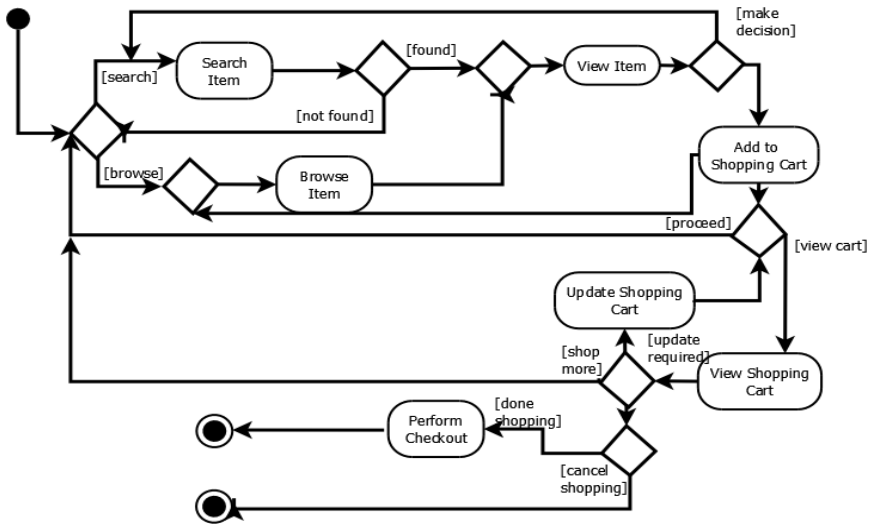


Figure 1 : An activity diagram for 'Customer does Online Shopping'

Activity diagrams have been used to generate test cases. A path is defined as a set of nodes and edges starting from the initial state to the final state. An independent path is one that traverses at least one new edge in the flow graph. Both true and false branches of all conditions are executed. A test case with reference to a path is one that traces an execution path from the initial state to the final state (Definition 2).

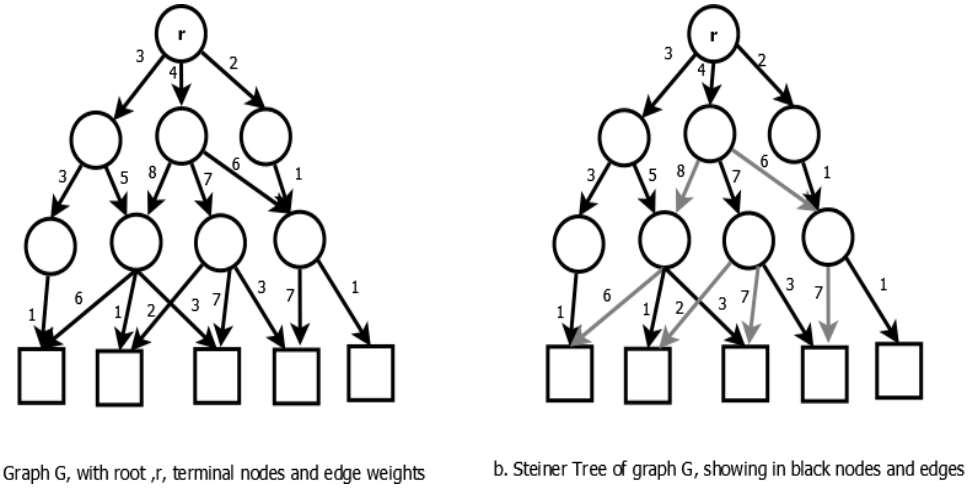
Definition 2: A test case , tc ∈ TC, in an activity diagram, AD, can be defined as an execution path from the initial state to the final state consisting of activities and transitions.

i.e. ∀ tc, where, tc ∈ TC, $tc = a_0 \rightarrow t_0 \rightarrow a_1 \rightarrow t_1 \rightarrow ..... \rightarrow t_n \rightarrow a_m$
where $a_i \in A$, $t_i \in T$,
a_0 is the initial state,
a_m is the final state.
TC is the set of test cases

2.3 Steiner Tree

Definition 3: Given a directed graph, G, a set of vertices, V, and edges, E, the goal of the directed Steiner tree problem is to find a minimum cost tree in a directed graph G = (V,E) that connects all terminals X to a given root r [9][10].

The Steiner Tree is distinguished from the Spanning Tree in that while a spanning tree spans all vertices of a given graph, a Steiner tree spans a given subset of vertices. In the case of the Steiner minimal tree problem, vertices are divided into two parts : terminals and non terminals. Terminals are the given vertices that must be included in the solution. Non terminals may be included if necessary to connect terminals. The cost of a Steiner tree is the total edge weight. In order to reduce the cost, non terminal vertices may be included.



a. Graph G, with root ,r, terminal nodes and edge weights

b. Steiner Tree of graph G, showing in black nodes and edges

Figure 2 : Example of a Graph G and its corresponding Steiner Tree

Thus, let G (V, E, w) be a directed graph, where given :
V, is the set of vertices
E, is the set of edges
r, is the root
a set L ⊂ V of terminals, and
w, is the weight of the edge(non-negative)

Steiner Minimal Trees are discussed in [9][10] and the algorithm (refer Algorithm 1) discussed in the work is given here for ease of reference. The objective is to compute a subset T ∈ E of edges such that there is an r - s path in T for each terminal s i.e. find a tree T with L ∈ V (T) so as to minimize w(T).

Algorithm 1 : Steiner Algorithm

Algorithm : MST-Steiner

Input: A graph G = (V, E, w) and a terminal set L ⊆ V
Output: A Steiner tree

1. Construct the metric closure G_L on the terminal set L
2. Find an MST T_L on G_L
3. T ← Φ
4. For each edge e = (u, v) ∈ E(TL) in a depth-first-search order of T_L do
   4.1 Find a shortest path P from u to v on G
   4.2 If P contains less than two vertices in T then
       Add P to T
   Else
       Let p_i and p_j be the first and the last vertices already in T
       Add subpaths from u to p_i and from p_j to v to T
5. Output T

For example, consider the graph given in Figure 1a with 'r' as the root. The terminal nodes are indicated as the square nodes and the weights for each of the edges is indicated. Figure 1b shows the Steiner tree with the subset of nodes and edges that constitute the minimal Steiner tree.

3. Generating Minimal Test Cases - The Approach

This work presents an approach to generate minimal regression test cases for a functionality which will give an indication of whether defects have been fixed. In this work, an activity diagram is used to elaborate functionality.

3.1 Preprocessing

The activity diagram is converted into a Control Flow Graph (CFG). Each activity in the activity diagram is a node in the CFG and a control flow in the activity diagram corresponds to an edge. In the case of loops, each loop is unfolded to a maximum of two iterations. The assumption is based on work in [19]. For fork join constructs that represent concurrency, it is considered as one node in this work. The weight for each edge is calculated using a measure e.g. distance measure. With reference to this work, edge calculation is based on the incoming and outgoing dependencies [6] as given below.

$$Weight(e) = (n_i)_{in} \text{ X } (n_j)_{out}$$

where $(n_i)_{in}$ is the number of incoming dependencies of node n_i and $(n_j)_{out}$ is the number of outgoing dependencies of node n_j and e is the edge connecting n_i and n_j.

3.2 Algorithm

Given the Control Flow Graph (CFG) elaborating a functionality, the next step is to define terminal nodes. The following rules are followed:

a. Consider as terminal the root node of the CFG.
b. Consider as terminals all the nodes that provide output of value to the user.
c. Add as terminals all nodes where change has been made.

Given that the technique is black-box, the first two rules ensure that all I/O nodes are included. The third rule incorporates change that has been done due to a bug fix or change in functionality. It is necessary that such nodes be added to the minimal set of test cases as they should be part of the regression test cases.

Algorithm 2: Generate

Algorithm : GS (Generate Test Cases)

Input: Activity diagram, A
A graph G = (V, E, w) and a terminal set L ⊂ V
Output: TC, a set of Test Cases

1. For each node in activity diagram, A do
   1.1. If node is an InitialNode, FinalNode, Activity, Decision node, convert into node in CFG.
   1.2 If node is a Fork, find corresponding Join and group all nodes inside the Fork-Join into one node.
2. For each edge in activity diagram, A do
   2.1 If edge forms a cycle(loop), unfold the loop upto two iterations adding nodes and edges to CFG.
   2.2 Else, add edge in CFG.
3. Calculate edge value using a measure.
4. Define the terminal edges.
5. Use Steiner Tree algorithm, MST-Steiner, to generate Steiner Tree, T.
6. Generate TC, the set of test cases by generating independent paths from the start node to end node of T.

TC, the minimal set of test cases is used to test the system.

4. Case Study

Consider as example an activity diagram for the 'Buy Beverage' functionality of the Vending Machine [6] in Figure 3a. The equivalent graph is shown in Figure 3b.

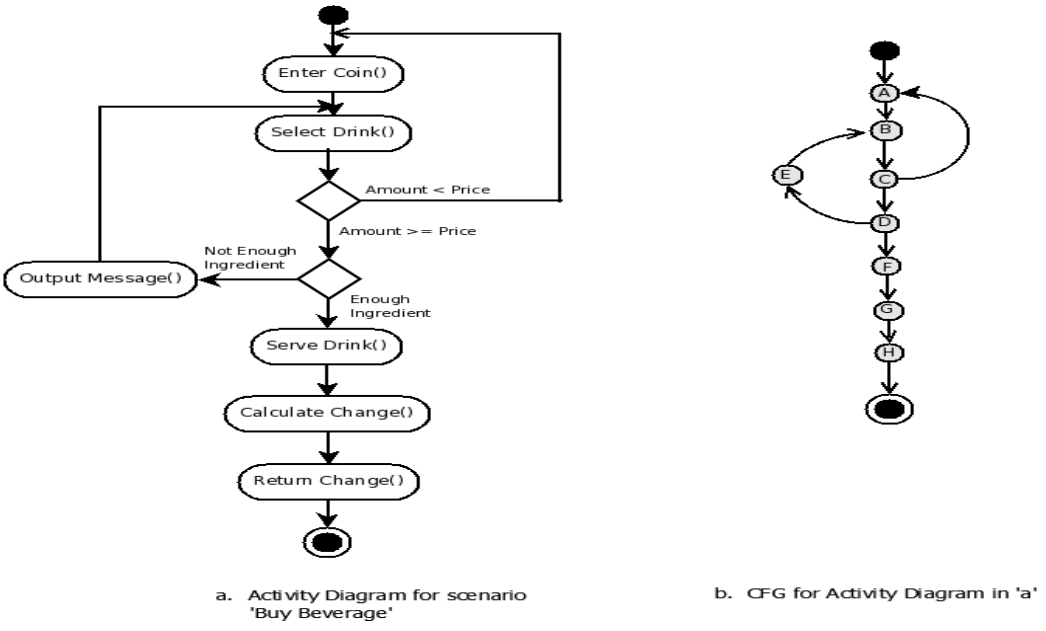


Figure 3 : Activity diagram for 'Buy Beverage' functionality and its corresponding CFG, G.

Given the CFG 'G', in Figure 3, the next step is to unfold cycles if any, and convert any fork-join constructs into a single node. The unfolded graph is shown in Figure 4a.

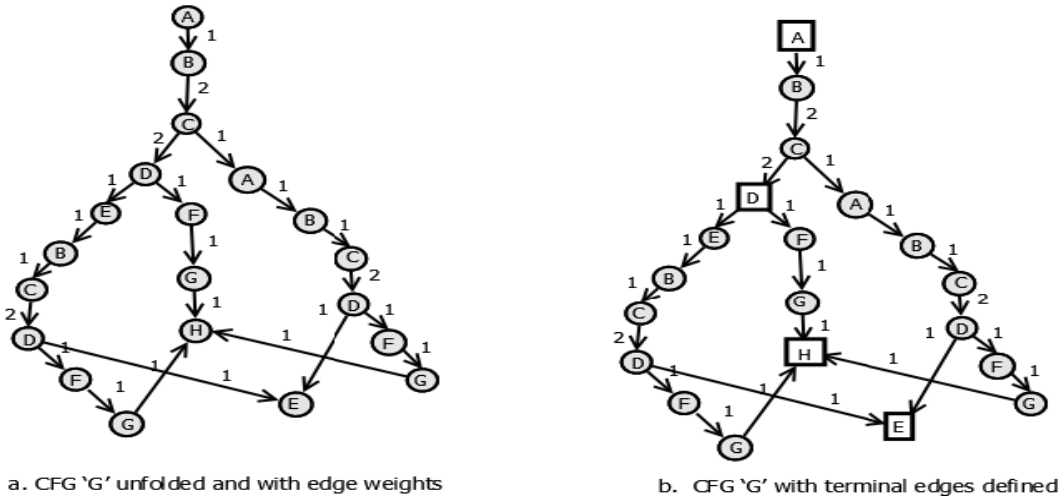


Figure 4 : CFG 'G' unfolded with edge weights and terminal edges defined

Figure 4 b shows the terminal edges defined. 'D' is considered as a terminal edge as a change was incorporated in the condition. 'D' is added to the set of terminals as it is an activity that must be exercised as part of testing. Applying the Steiner minimum tree algorithm, it is found that minimal path to the terminal activities are the following :

1. A – B – C – D – F – G – H
2. A - B – C – A – B – C – D - E

Thus, two test cases form the minimal test case to test this functionality for regression testing. The two test cases provide initial information to the test engineers whether the bug fix was successful. In case the test cases do not pass, then testing entire functionality can be avoided and the artifact reported to the development team for debugging. Effort can be applied to other functionality that need to be tested and results given to the test engineers.

5. Conclusion and Future Work

In this work, a black box approach for generating test cases for regression testing has been proposed. The objective was to create minimal regression test set that serve as an indicator whether bug fixing has been successful. UML has been used to model requirements. Each functionality is elaborated using activity diagrams. Given a set of terminal nodes, the Steiner tree algorithm was used to generate the minimal regression test set. The approach is advantageous as the nodes that have a change can be added to the set of terminals and hence included in the test set.

The approach has been tested on individual functionality elaborated through activity diagrams. This work however imposed certain constraints. First, a loop was unfolded twice to contain the size of the graph. Secondly, a fork-join construct which represents concurrent activity is considered as a single node. The size of the case studies undertaken does not reflect the scale of actual software systems. Future work involves exploring further the following issues : a. build case studies to understand the efficacy of the technique on large functionality; b. explore the use of other parameters like cost, risk, etc to calculate weight of edges; c. mechanism for testing of fork-join constructs and d. development of a tool to support the technique.

6. References

[1] S. Elbaum, A. Malishevsky and G. Rothermel. Test Case Prioritization: A family of empirical studies. IEEE Transactions on Software Engineering, 28(2),159-182, February 2002.

[2] Zheng Li, Mark Harman, and Robert M. Hierons. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering, 33(4), April 2007.

[3] Bogdan Korel and George Koutsogiannakis. Experimental comparison of code-based and model-based test prioritization. In IEEE International Conference on Software Testing Verification and Validation Workshops, 2009.

[4] OMG. Unified modeling language (UML) Superstructure Specification, version 2.1. Technical report.

[5] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.

[6] Sapna P.G. and Hrushikesha Mohanty. Prioritization of scenarios based on UML activity diagrams. In 1st International Conference on Computational Intelligence, Communication Systems and Networks(CICSYN 2009), pages 271-276. IEEE Computer Society, 2009.

[7] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. Software Engineering,27(10):929-948, 2001.

[8] G. Rothermel and M. Harrold. A safe, efficient regression test selection technique. ACM Transactions on Software Engineering and Methodology, 6(2):173-210,April 1997.

[9] T. Rothvo. Directed Steiner tree and the Lasserre hierarchy. CoRR, abs/1111.54-73, 2011.

[10] Bang Ye Wu: A simple approximation algorithm for the internal Steiner minimum tree. CoRR abs/1307.3822, 2013.

[11] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, JRipples: A tool for program comprehension during incremental change, in Proceedings of the 13th International Workshop on Program Comprehension (IWPC 05), pp. 149-151, May 2005.

[12] Baradhi, G., Mansour, N., A Comparative Study of Five Regression Testing Algorithms, Proceedings of the IEEE Australian Software Engineering Conference, pp. 174-182, 1997.

[13] Korel, B., AI-Yami, A.,Automated Regression Test Generation, Proceedings of IEEE International Symposium on Software Testing and Analysis, pp. 143-152, 1998.

[14]L.C. Briand, Y. Labiche, G. Soccar. Automating Impact Analysis and Regression Test Selection based on UML Designs. Proceedings of the International Conference on Software Maintenance (ICSM 2002), IEEE Computer Soceity, pp.1-10, 2002.

[15] Yanping Chen, Robert L. Probert, A Risk-based Regression Test Selection Strategy, in the Proceeding of the 14th IEEE International Symposium on Software Reliability Engineering (ISSRE 2003), Fast Abstract, pp. 305 - 306, November, 2003.

[16] Jayaprakash Krishnan, Gunasekar Subramani and Sapna P G.. Article: Development of a Change Impact Model for Regression Testing using UML Diagrams. IJCA Proceedings on International Conference on Distributed Computing and Internet Technology 2014 ICDCIT-2014:31-36, December 2013.

[17] Sapna, P.G.; Mohanty, H., "Automated Scenario Generation Based on UML Activity Diagrams," Information Technology, 2008. ICIT '08. International Conference on , vol., no., pp.209,214, 17-20 Dec. 2008.

[18] W. Sinzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang. Generating Test Cases from UML Activity Diagram based on Gray-Box Method. In Proceedings of the 11th Asia-Pacific Software Engineering Conference(APSEC). IEEE Computer Society, November 2004.